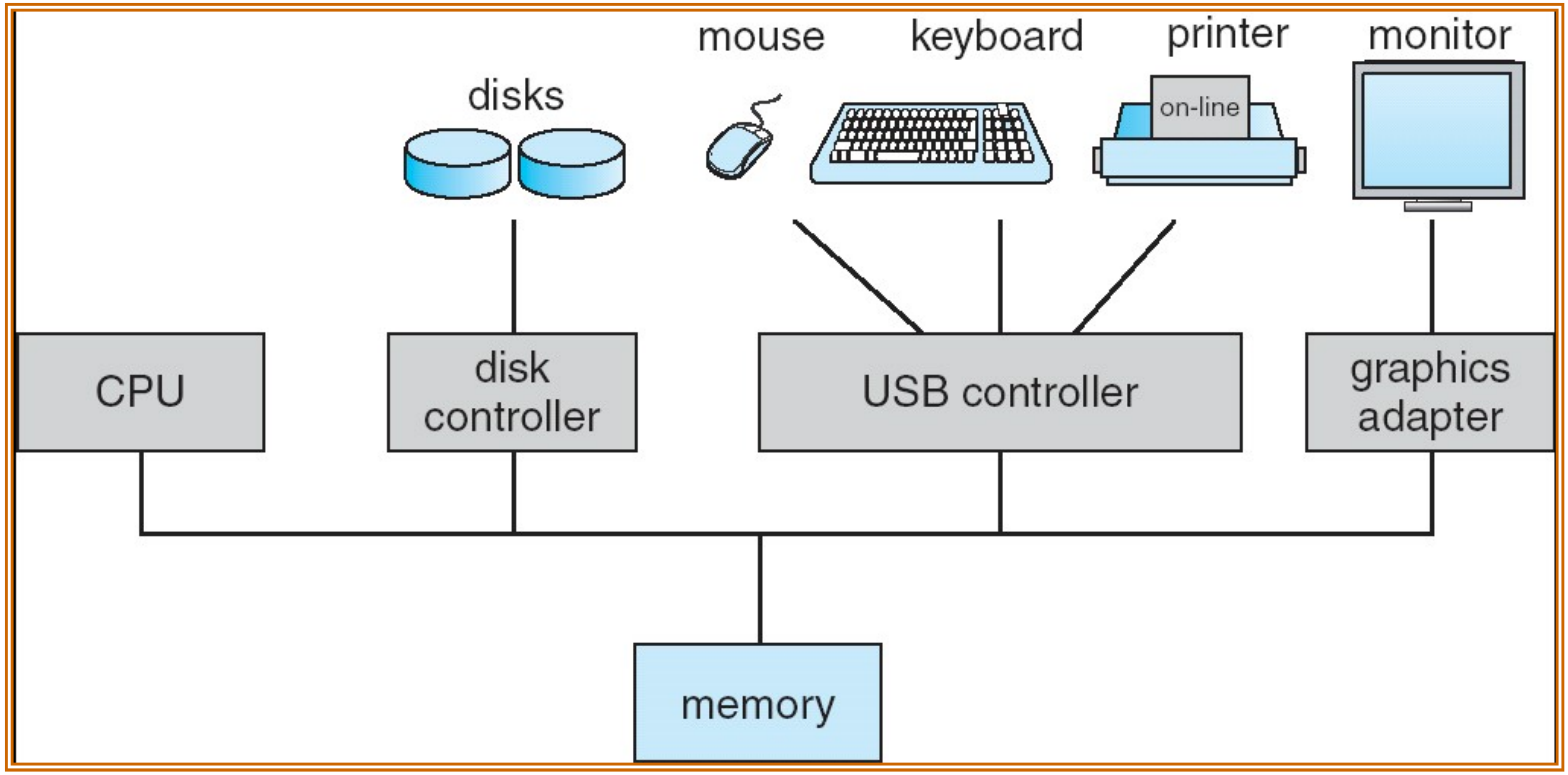
Università degli Studi di Parma
Dipartimento di Ingegneria e Architettura

Richiami di Sistemi Operativi

ed evoluzione delle tecnologie

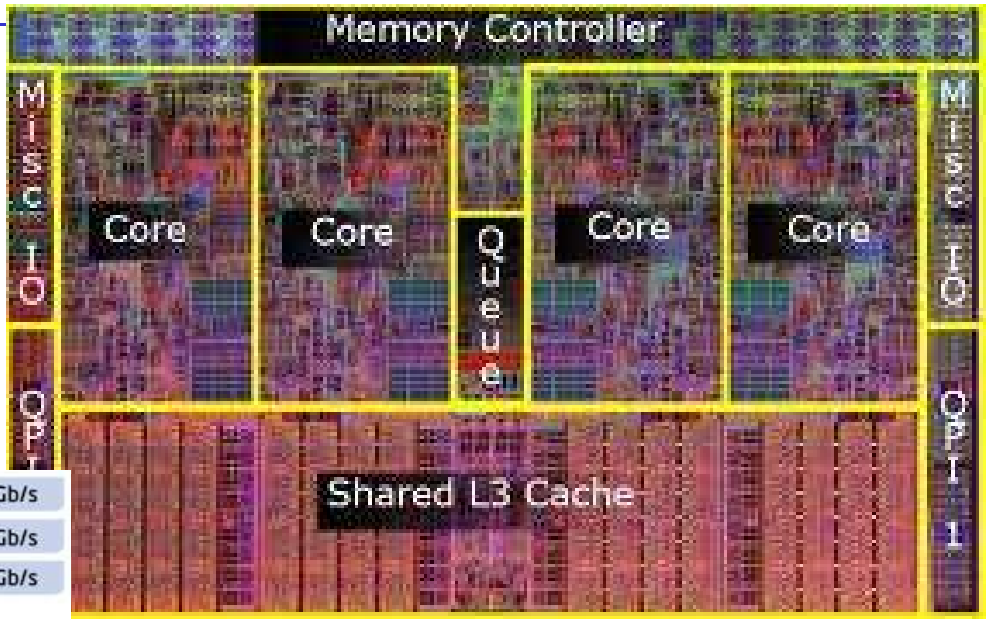
Organizzazione di un sistema di elaborazione

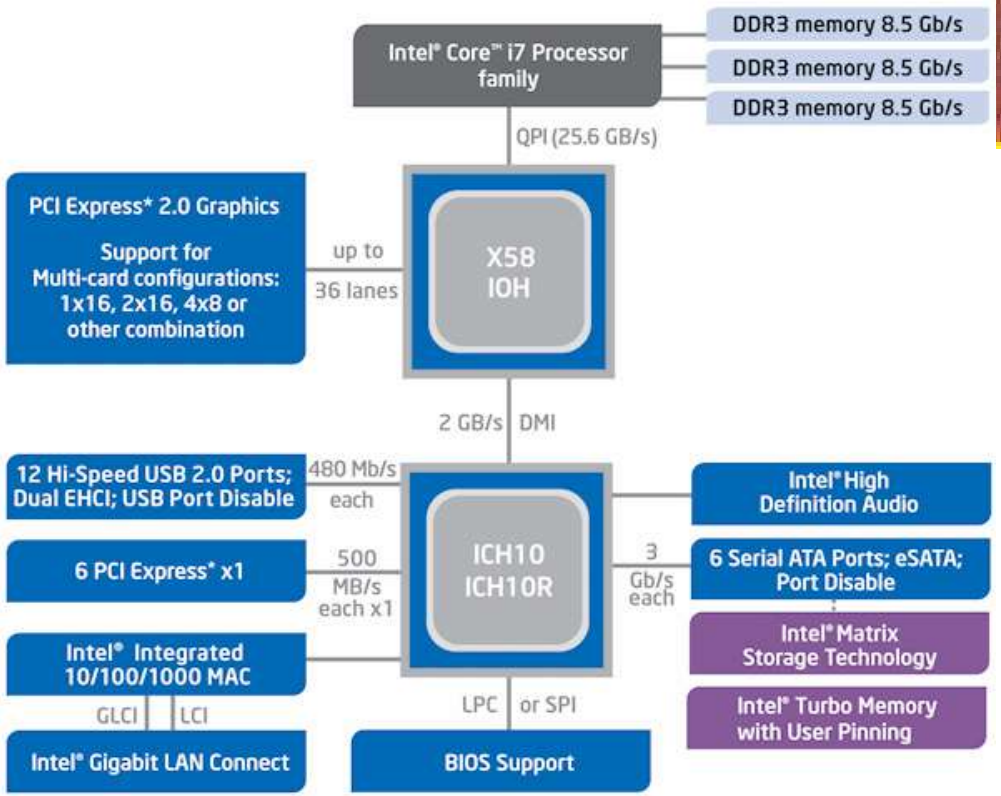
- Una o più CPU e controllori di dispositivi collegati mediante un bus comune che fornisce accesso alla memoria condivisa
- CPU e controllori eseguono in modo concorrente e competono per i cicli di accesso alla memoria



Dentro una CPU recente

- nel nostro portatile, o forse nel nostro telefono:



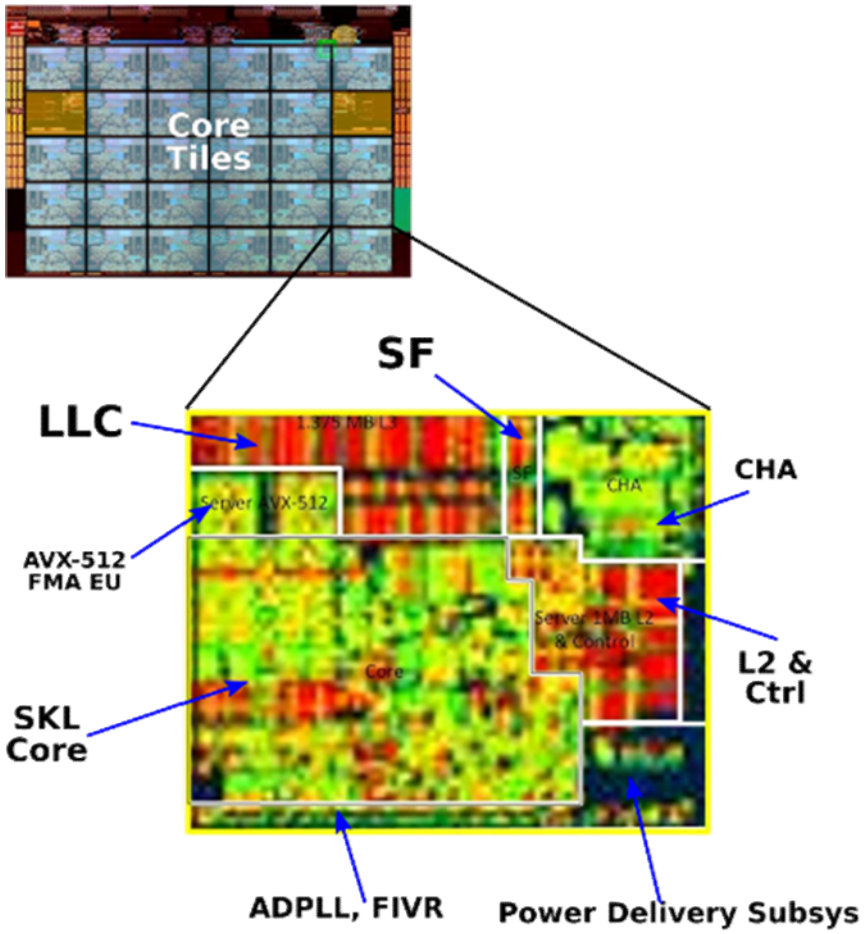


Problemi?

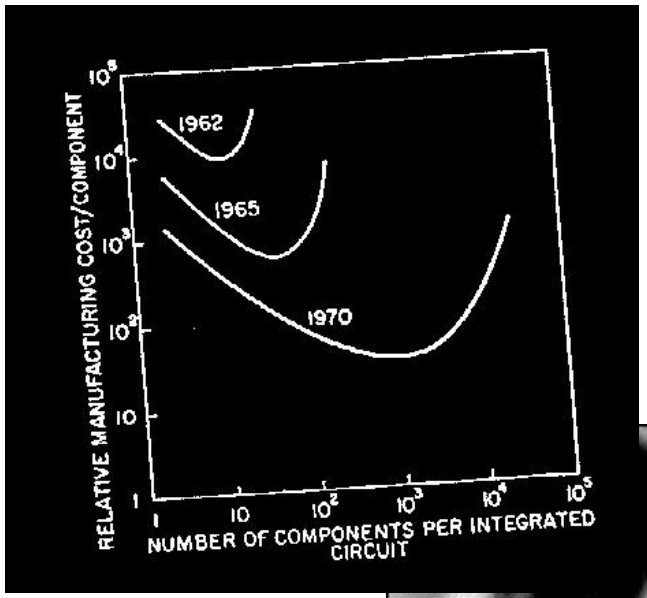
- Complessità
  - Le prestazioni elevate sono ottenute con architetture sempre più complesse
- Eterogeneità
  - Dal supercalcolatore al microcontrollore per la lavatrice o per il sensore IoT
  - Oggi: relè = IED (Intelligent Electrical Device)
- Variabilità e fattore di scala:
  - Da 2 a 6 ordini di grandezza per tutti i parametri
- Attualizzazione della legge di Moore
  - Chip multicore e multithreaded ➔ parallelismo esplicito on-chip, necessità di nuovi paradigmi di programmazione

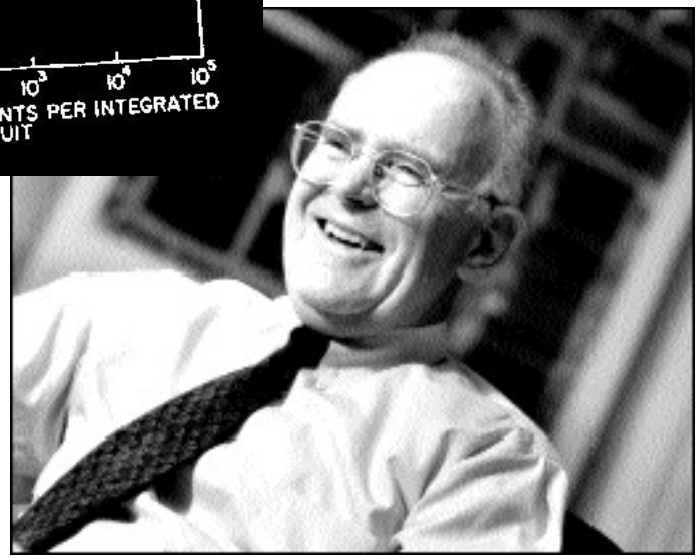
Verso architetture parellele pervasive

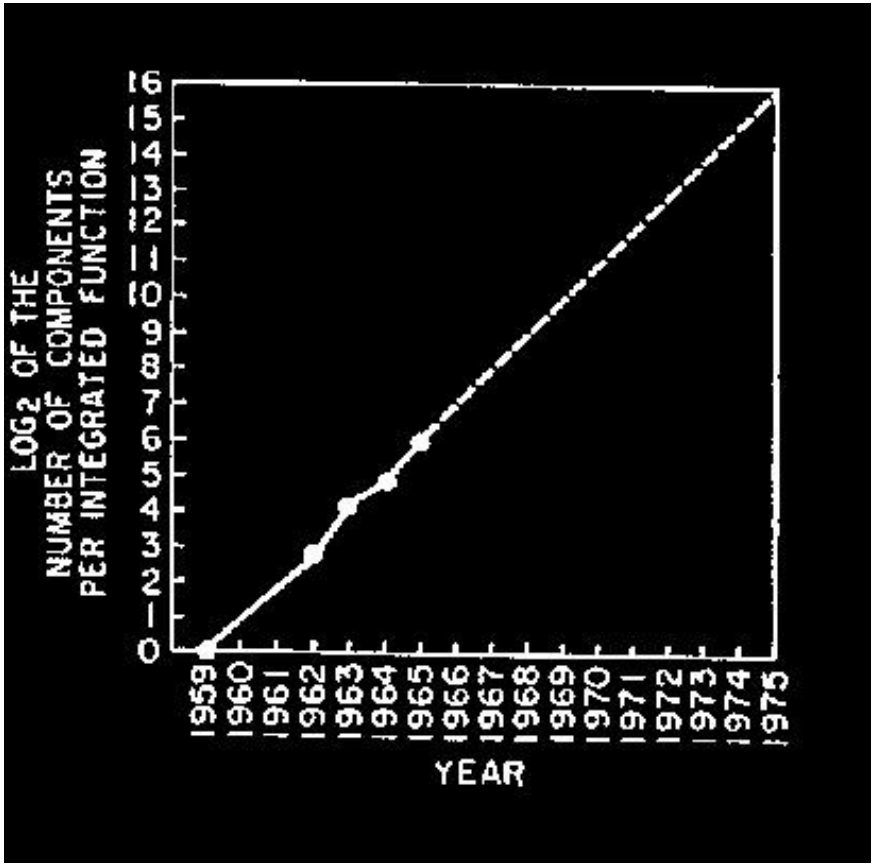
- Intel Xeon W-3175-X, 2019 (Skylake 2017)
- 14 nm, die 694 mm2
- 28 core, 56 thread
- 3.10 GHz
- cache on-chip: L2 28 MB, L3 condivisa 38.5 MB



Da dove viene questa complessità?



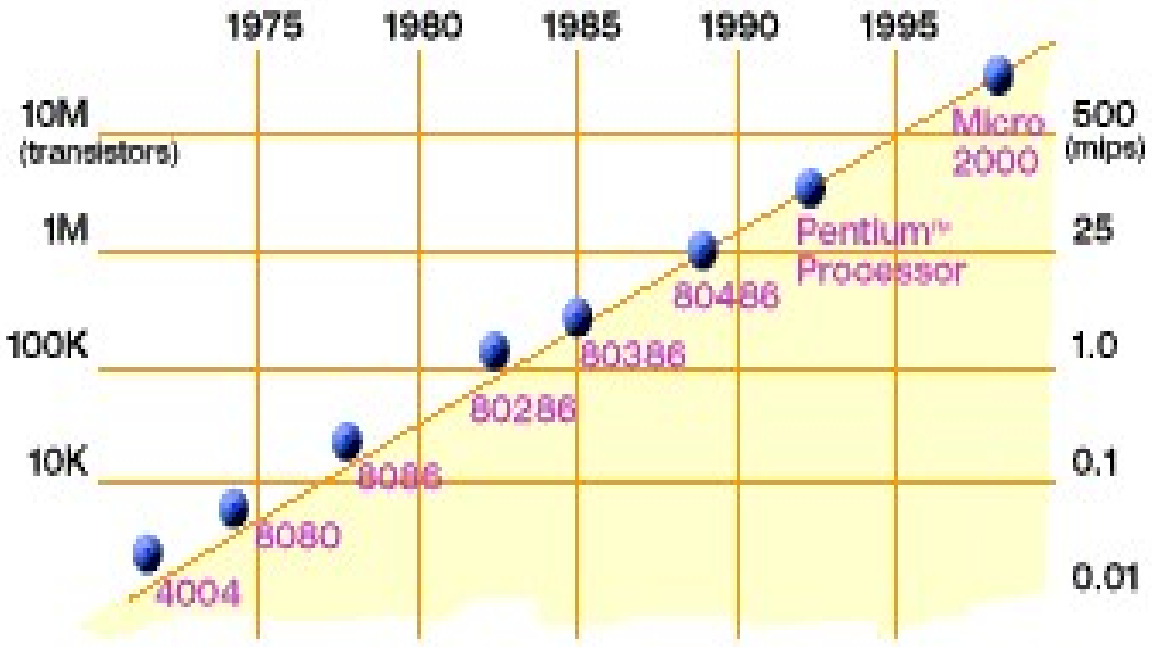




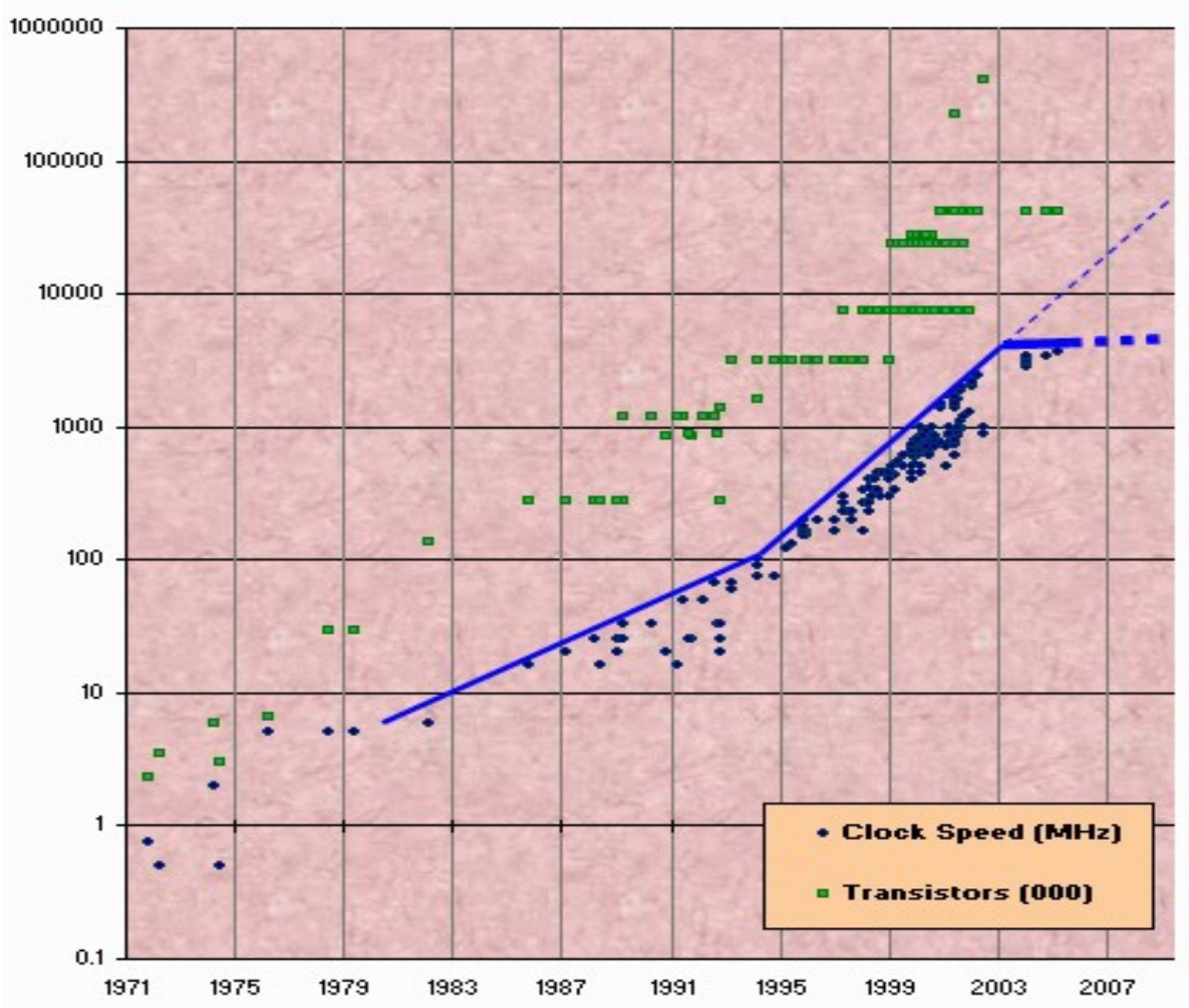
- "Cramming More Components onto Integrated Circuits"
  Gordon Moore, Electronics, 1965

Legge di Moore

- Una previsione empirica che si è avverata: il numero di transistori per chip è circa raddoppiato ogni 18 mesi
- Nel corso delle generazioni successive i microprocessori sono diventati più piccoli e più potenti → più intelligenza in meno spazio!

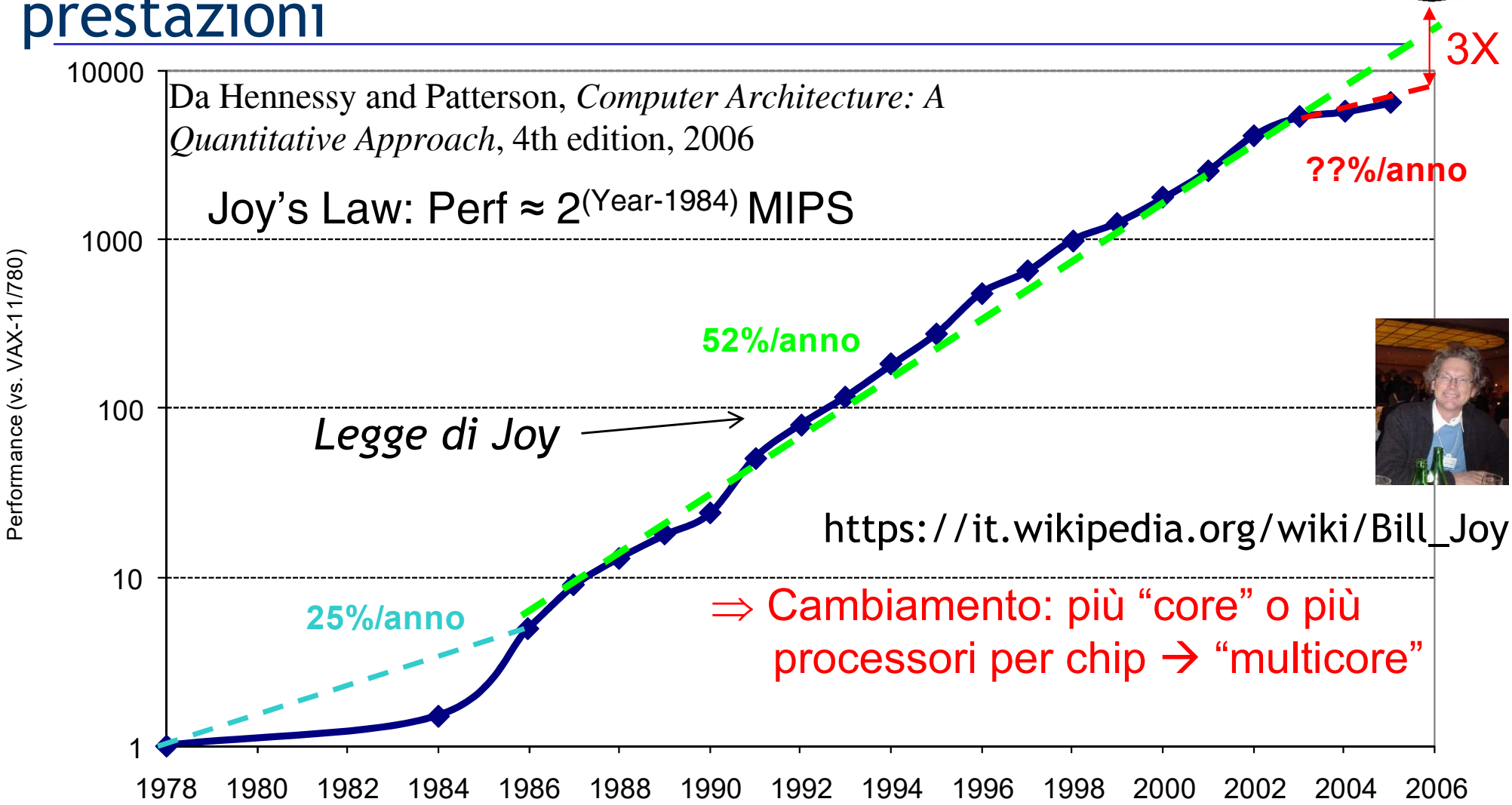


Legge di Moore, uhm



- Il numero di transistori continua a salire
- La frequenza di clock resta quasi costante

Problema: dal 2002 rallenta la crescita delle prestazioni



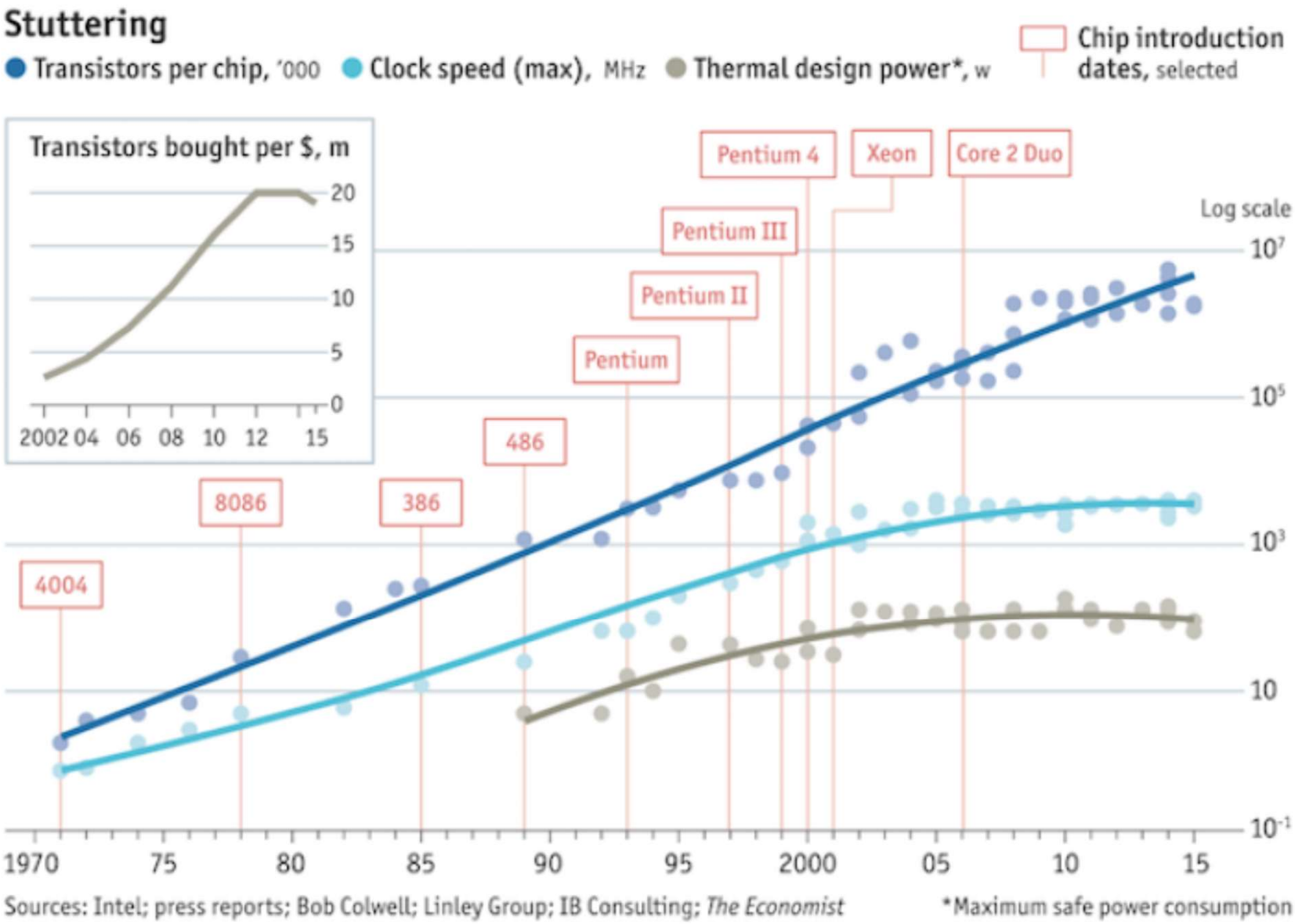
https://it.wikipedia.org/wiki/Bill_Joy

⇒ Cambiamento: più "core" o più processori per chip → "multicore"

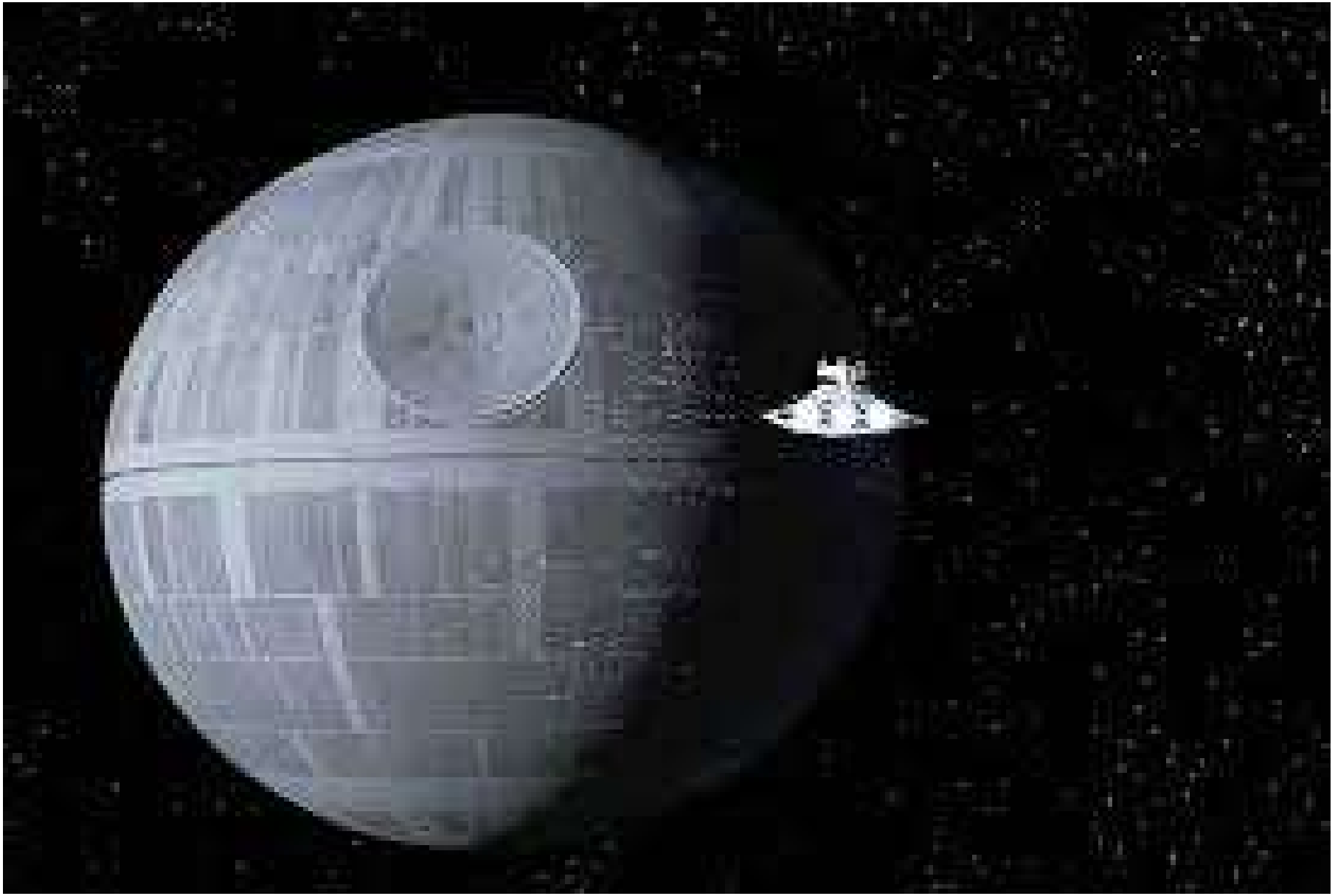
- VAX : 25%/anno 1978 - 1986
- RISC + x86: 52%/anno 1986 - 2002
- RISC + x86: ??%/anno 2002 - adesso

https://en.wikipedia.org/wiki/Joy's_law_(computing)

«Dennard scaling is over» (Bob Colwell, former Intel chief designer)

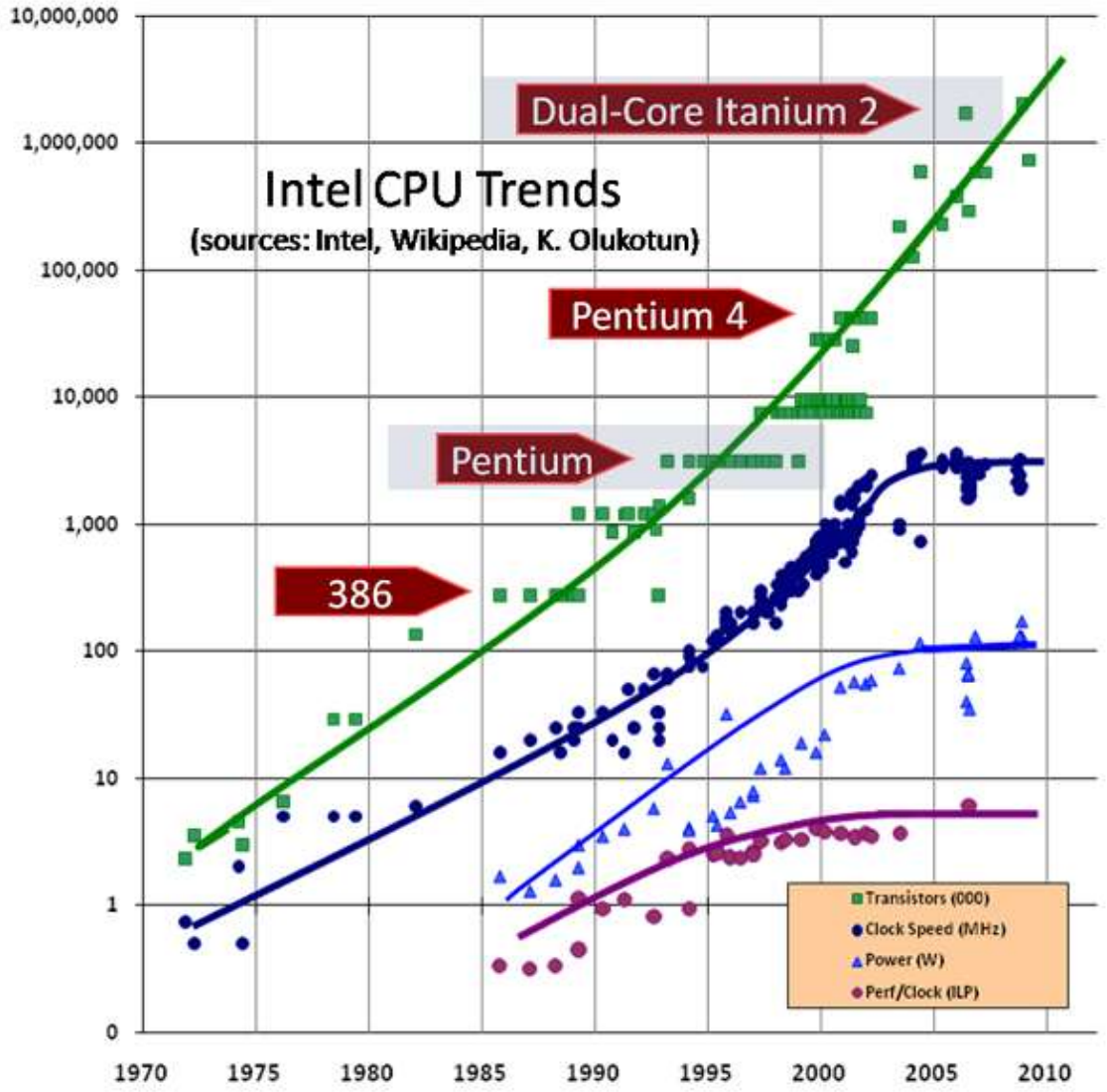


Dark silicon



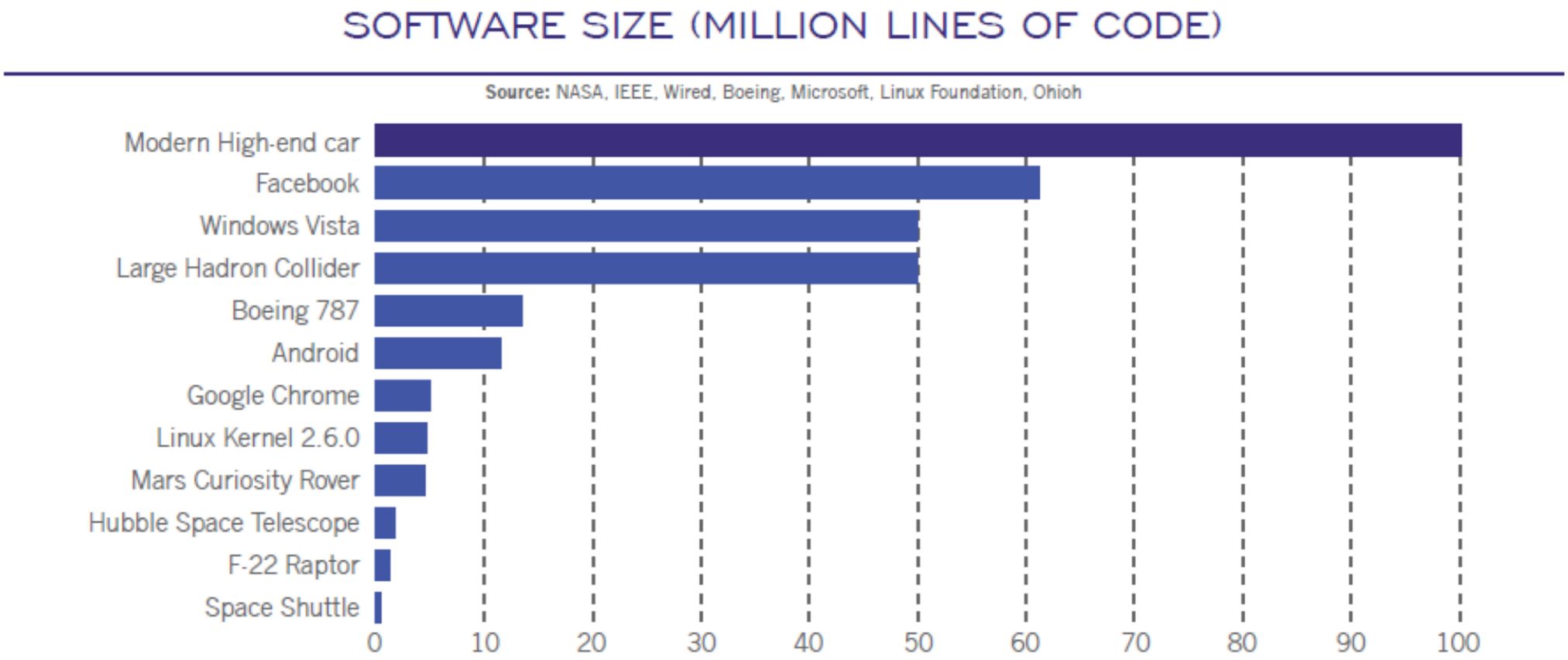
The free lunch is over

- «A fundamental turn toward concurrency in software» (H. Sutton, Dr. Dobb's J., 2005 & 2009)



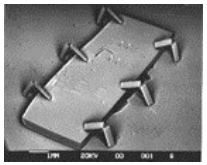
Inoltre, programmi sempre più grandi e complessi

- https://www.informationisbeautiful.net/visualizations/million-lines-of-code/
- https://medium.com/next-level-german-engineering/porsche-future-of-code-526eb3de3bbe



Come gestire la complessità?

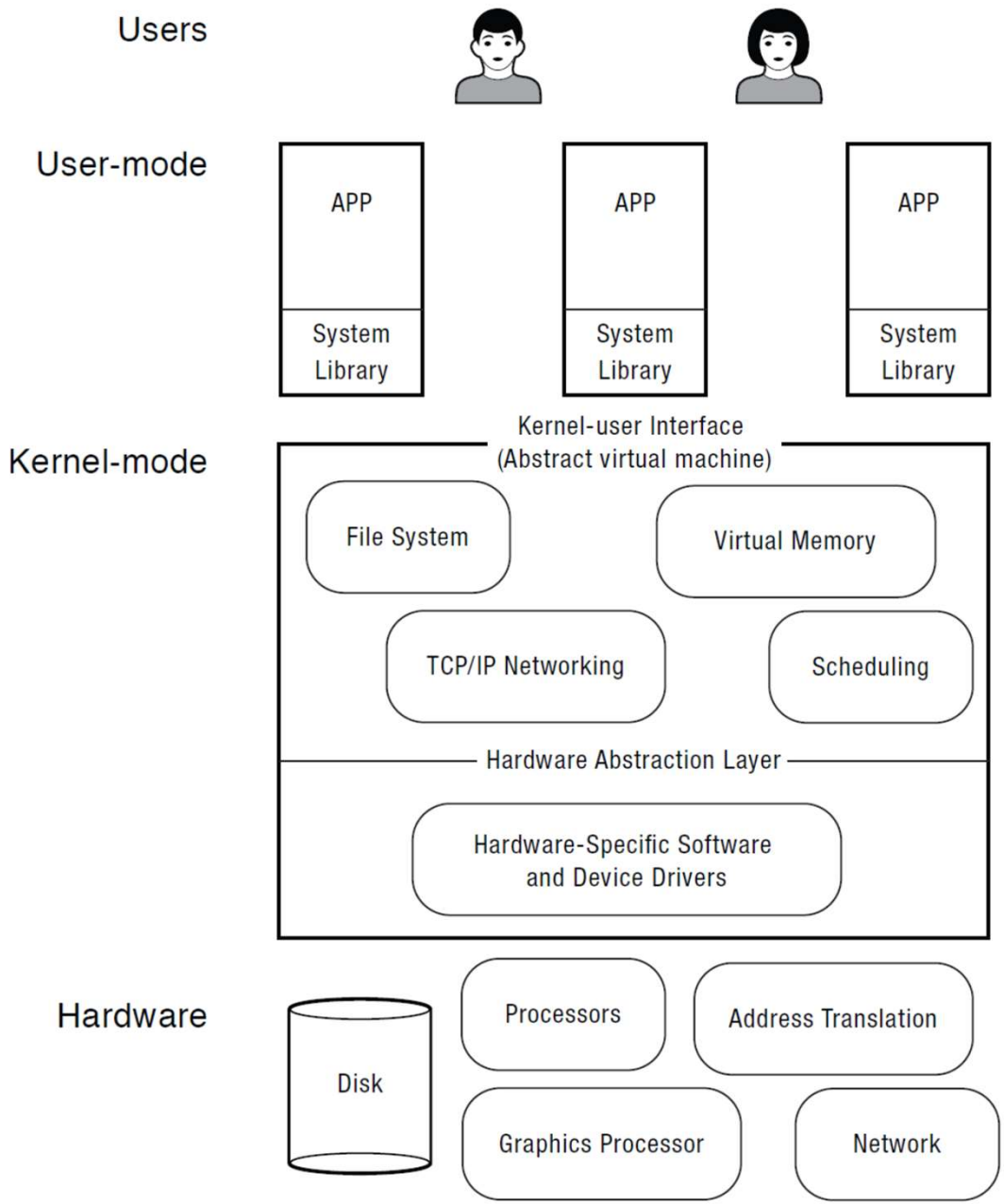
- Ogni computer è diverso:
  - CPU (intel i7, PowerPC, ARM, etc.)
  - Memoria principale e di massa disponibili
  - Dispositivi (mouse, tastiere, touch interface, sensori, fotocamere, interfacce biometriche, ...)
  - Ambiente di rete (cablata, DSL, wireless, firewall,...)




- $\Rightarrow$ Il SO soddisfa esigenze essenziali:
  - Al programmatore non è richiesto di scrivere un unico programma per l'esecuzione di più attività indipendenti
  - Non è necessario modificare i programmi in funzione di ogni tipo di componente hardware
  - Un programma malfunzionante non provoca il crash dell'intero sistema
  - Ai programmi utente non è richiesto di avere accesso a tutto l'hardware




Struttura di un SO general-purpose



Complessità hw e sw

- Più programmi in esecuzione che competono per le risorse del sistema di elaborazione? Possibile?

- Se, come è, l'uso efficiente dell'hw complesso implica la esecuzione «simultanea» di più programmi o più attività computazionali, ci sono implicazioni per la correttezza della esecuzione?
- Altrimenti, a che servono i core multipli?

- Risposta in breve: sì

La sfida della complessità

- Le attuali applicazioni software sono costituite da moduli software eterogenei, in esecuzione su sistemi di elaborazione diversi, in ambienti «challenging»:
  - architetture hardware diverse ed eterogenee
  - altre applicazioni che competono per le risorse
  - malfunzionamenti imprevisti
  - possibili attacchi di diverso tipo
- E' impossibile verificare/collaudare un programma per tutti gli ambienti in cui potrà essere eseguito e per le combinazioni di componenti sw e dispositivi hw
  - è scontato che ci potranno essere «bug»
  - è essenziale che i bug non siano gravi e siano invece «confinati»
  - è necessario costruire applicazioni corrette «by design»

Obiettivi del sistema operativo

- La nascita dei SO è stata guidata da due requisiti fondamentali:
  - Affidabilità / robustezza
  - Efficienza
- A questi si è rapidamente aggiunto il requisito:
  - Semplicità di utilizzo del sistema di elaborazione

- Nell'informatica degli ultimi decenni si sono aggiunti due ulteriori requisiti:
  - Portabilità
  - Sicurezza

L'astrazione *Macchina Virtuale*

Applicazioni Sw

Interfaccia *Virtual Machine*

Sistema Operativo

Interfaccia *Physical Machine*

Hardware

- Un'idea classica dei SO:
- Trasforma funzionalità e caratteristiche hw/sw di basso livello in servizi orientati al programmatore e ottimizzati per facilità di utilizzo, efficienza, sicurezza, affidabilità, etc.
- Si applica in ogni area dei SO (file system, memoria virtuale, gestione rete, scheduling, etc.): l'interfaccia hardware è complessa, alla applicazione è offerta un'astrazione di più alto livello

Macchine Virtuali

- Emulazione software di una macchina astratta
  - Fornisce ai programmi l'*illusione* di eseguire su una macchina dedicata
  - Rende (come se) presenti funzionalità non offerte dall'hw
  - Consente porting programmi da un hw/OS ad un altro
- Vantaggi: semplicità, robustezza, efficienza

- Due tipi di macchine virtuali:
  - Macchina virtuale "Processo": consente l'esecuzione di un singolo programma; astrazione realizzata dal SO
  - Macchina virtuale "Sistema": consente l'esecuzione di un intero SO e delle sue applicazioni (es. VMware, Xen, …) in un contenitore protetto

Macchine Virtuali “Processo”

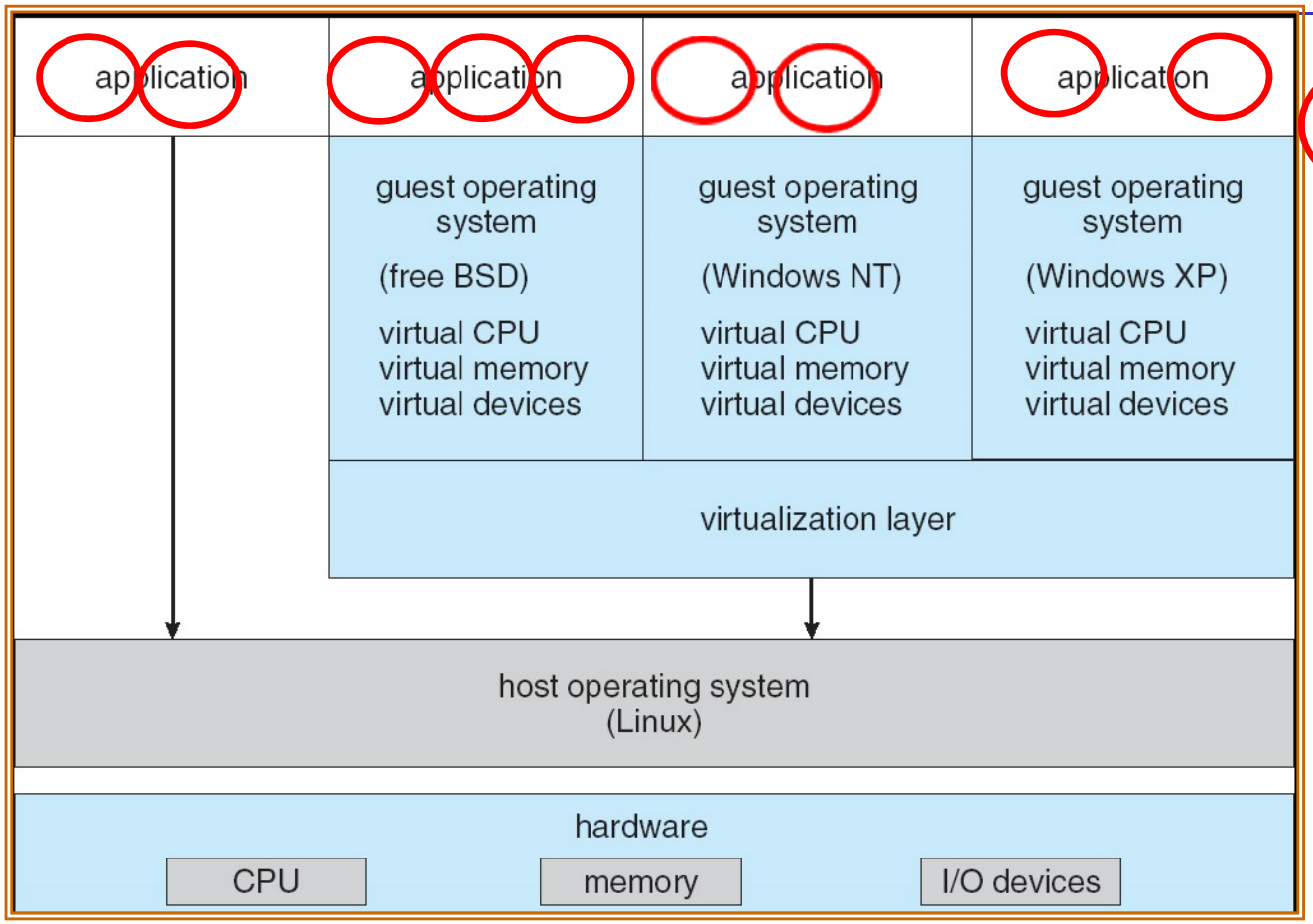
- Semplicità di programmazione:
  - Ogni programma in esecuzione ritiene di possedere tutte le risorse (memoria, CPU, dispositivi)
  - Tutti i dispositivi presentano la medesima interfaccia di alto livello
  - Le interfacce dei dispositivi appaiono più potenti dell’hw reale (es. scheda Ethernet vs. servizio di scambio di messaggi; display bitmapped vs. sistema a finestre)

Macchine Virtuali “Processo”

- Robustezza:
- Isolamento di guasti e malfunzionamenti
  - I programmi in esecuzione non possono influenzarsi direttamente l’uno con l’altro
  - Errori di programmazione non bloccano l’intero elaboratore
- Protezione e portabilità → concetti perseguiti ad esempio da Java

- Efficienza:
- Ottimizzazione nella allocazione delle risorse fisiche, se overhead di virtualizzazione limitato

Macchine Virtuali «Sistema»

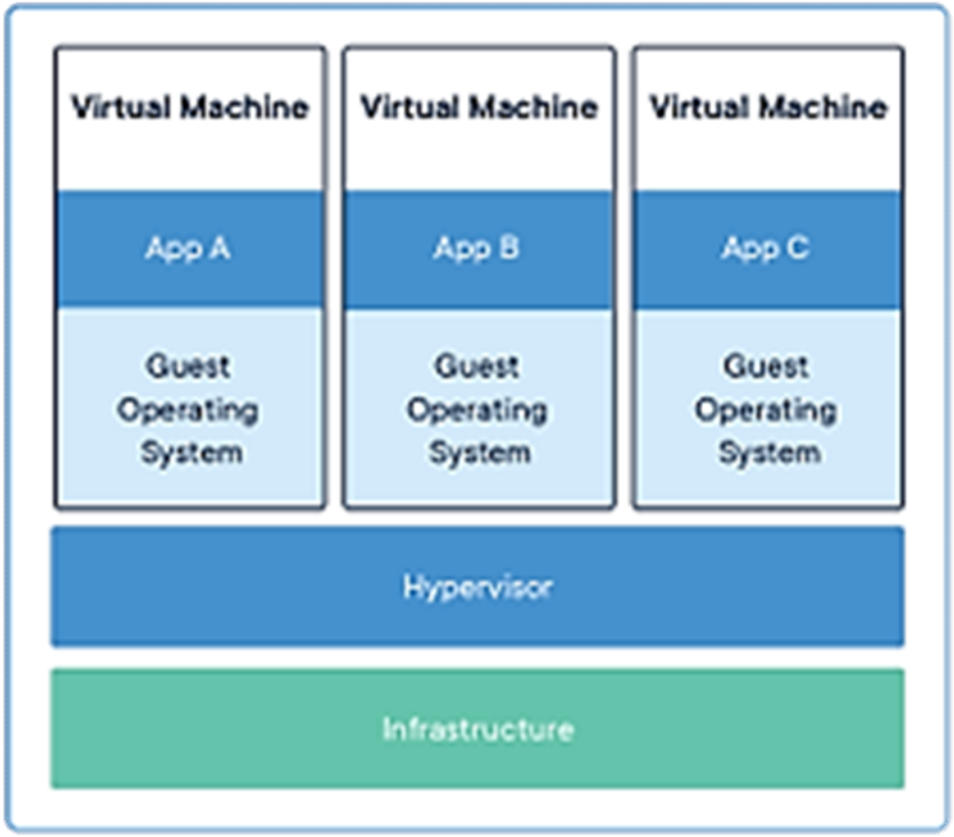


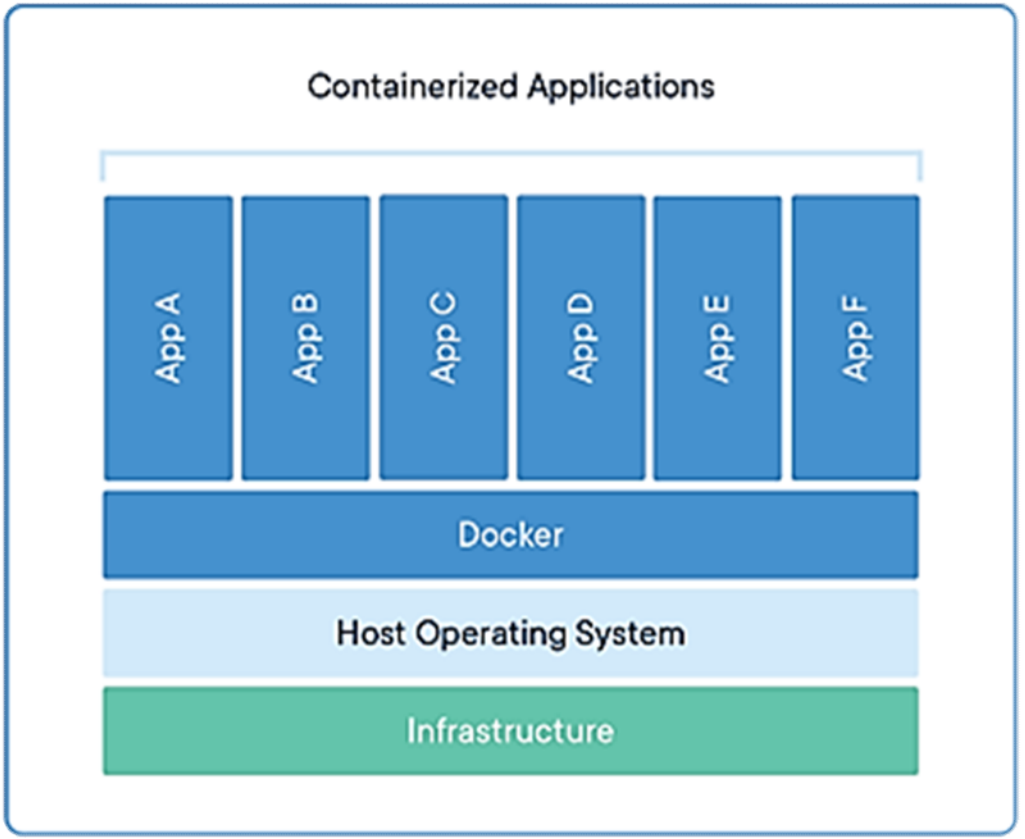
each guest or host OS can actually support multiple applications simultaneously

- Per sviluppare nuove applicazioni sw e SO, sandboxing
- Per condividere risorse utilizzate in modo variabile
- Es. VMware: anche al cuore di molti sistemi UniPR!

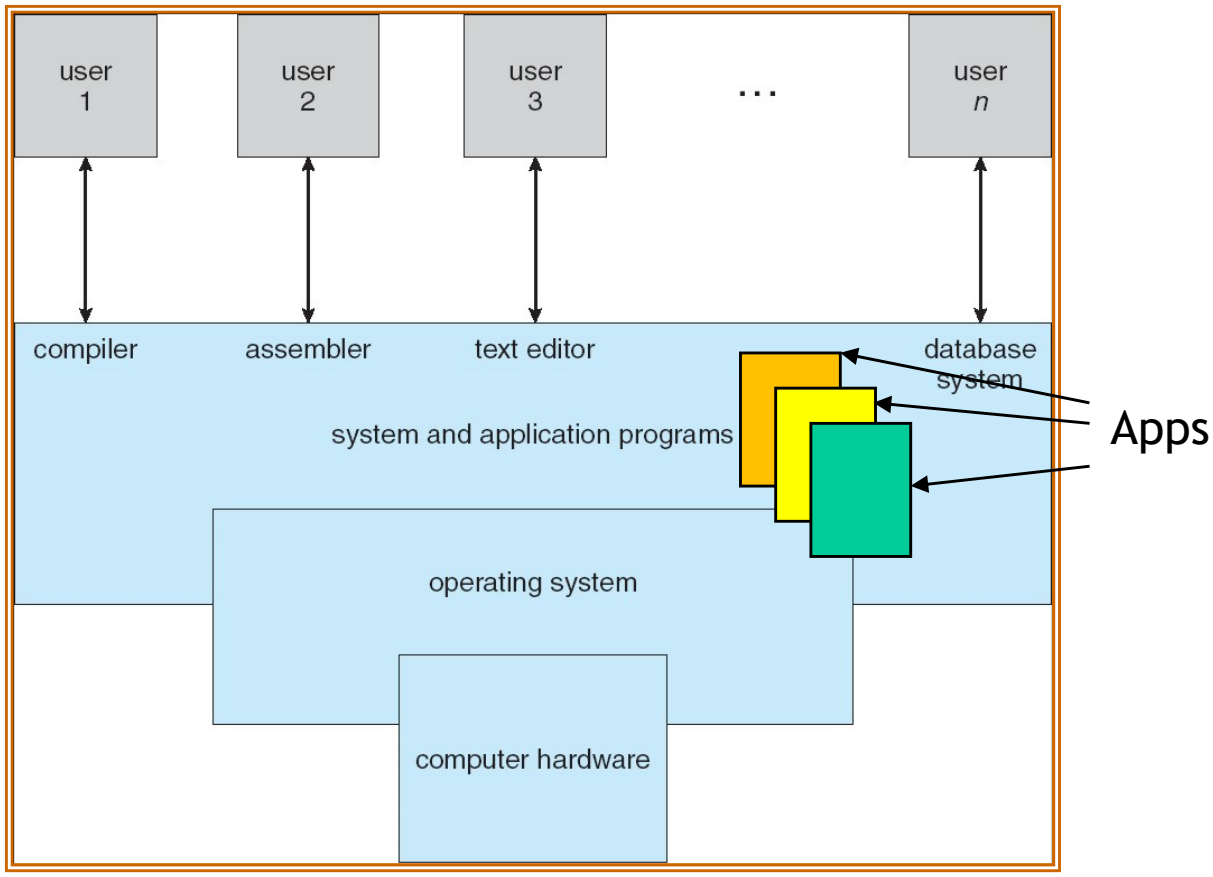
Virtualizzazione SO

- Approccio standard nei sistemi cloud
- Strati di protezione per gestire complessità ed eterogeneità





I componenti di un sistema di elaborazione



Un SO realizza una macchina virtuale più agevole e più sicura da programmare ed utilizzare rispetto all'hw sottostante.

Definizione di Sistema Operativo

- V. Cap.1 Silberschatz o Anderson!
- “Il SO è un programma che agisce come intermediario tra l’utente e gli elementi fisici di un calcolatore. Il suo scopo è fornire un ambiente nel quale l’utente possa eseguire programmi in modo agevole ed efficiente.”

- Pragmaticamente:
- Ciò che fornisce il venditore quando si ordina il SO?
- L’unico programma *sempre* in esecuzione nel calcolatore è il *kernel* o *nucleo*.
- Tutto il resto sono *programmi di sistema* (forniti con il SO) o programmi applicativi.

Sistema Operativo

- Fornisce un'astrazione di macchina virtuale gestendo e mascherando hardware complesso ed eterogeneo
- Coordina le risorse e assicura protezione mutua tra gli utenti e/o le applicazioni nell'accesso alle risorse condivise
- Semplifica lo sviluppo delle applicazioni fornendo servizi standardizzati
- Integra meccanismi e strategie per contenere l'effetto di guasti ed errori:
  - fault containment
  - fault tolerance
  - fault recovery

Principi sistemistici dei SO

- SO come *illusionista*:
  - Supera le limitazioni dell'hardware
  - Fornisce l'illusione di una macchina dedicata con memoria e processori illimitati
- SO come *governo o arbitro*:
  - Protegge gli utenti tra loro
  - Alloca le risorse in modo efficiente ed equo
- SO come *sistema complesso*:
  - Costante tensione tra semplicità e funzionalità o prestazioni
- SO come *insegnante di storia*:
  - Impara dal passato
  - Adattati man mano che cambiano i tradeoff hardware
- → Obiettivo dei SO è conciliare due aspetti fondamentali:
  - *Astrazione*
  - Concretezza, *efficienza*

Storia dei sistemi operativi

- Come eravamo

Alcune idee centrali sui Sistemi Operativi

- Alcune idee centrali sui sistemi operativi:
  - Meccanismi vs. politiche
  - Monoprogrammazione vs. multiprogrammazione
  - Multiprogrammazione vs. time sharing
  - Time sharing vs. elaborazione batch

Meccanismi vs. politiche

- Le *politiche* sono i modi in cui vengono scelte le attività da eseguire (in SO / CS ma non solo!)
- I *meccanismi* sono realizzazioni concrete che rendono possibili e attuano le politiche, e possono dipendere dall'hw sottostante (es: attribuzione di risorsa a processo con politica FCFS)
- *Separazione di meccanismi e politiche:*
- *I meccanismi non devono dettare le politiche attuate dal SO* (quali operazioni eseguire, quali risorse allocare)
  - $\Rightarrow$ sarebbe sbagliato «cablare» le politiche come meccanismi entro il SO: le politiche che ci interessano cambiano, nel tempo e nei diversi sistemi!
- Principio di progettazione fondamentale dell'informatica

Altre idee

- *Multiprogrammazione vs. time sharing*
- Due concetti ortogonali:
  - la *multiprogrammazione* riguarda l'architettura del SO e come realizza l'esecuzione dei programmi utente
  - il *time sharing* è una modalità di gestione dei programmi utente, che si alternano nel tempo
  - tuttavia, senza multiprogrammazione il time sharing sarebbe altamente inefficiente

- *Time sharing vs. elaborazione batch*
- Termini usati per distinguere modalità di interazione utente, interattiva o non, con i programmi in esecuzione

Monoprogrammazione vs. multiprogrammazione

- Il passaggio chiave nella nascita dei moderni SO
- Una lunga storia, in breve!

«Numbers everyone* should know»

- Da una famosa presentazione di Jeff Deans (Google):

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns
waiting for user keystroke	100,000,000 ns

  - meglio sovrapporre elaborazione all'I/O !

Il percorso al concetto di multiprogrammazione

- Indipendenza delle attività di I/O, tra loro e rispetto al programma in esecuzione
- Parallelizzazione della esecuzione del programma con le attività di I/O
- Sovrapposizione dell'esecuzione del programma con le attività di I/O di *altri* programmi
- Serve il concetto di interruzione!

Multiprogrammazione

- Per un efficiente uso delle risorse occorre avere *più programmi presenti contemporaneamente in memoria*, potendo così sovrapporre elaborazione ed I/O di programmi diversi
- Pro: efficienza, possibilità di politiche di gestione
- Contro: complessità, problemi di sicurezza

Multiprogrammazione

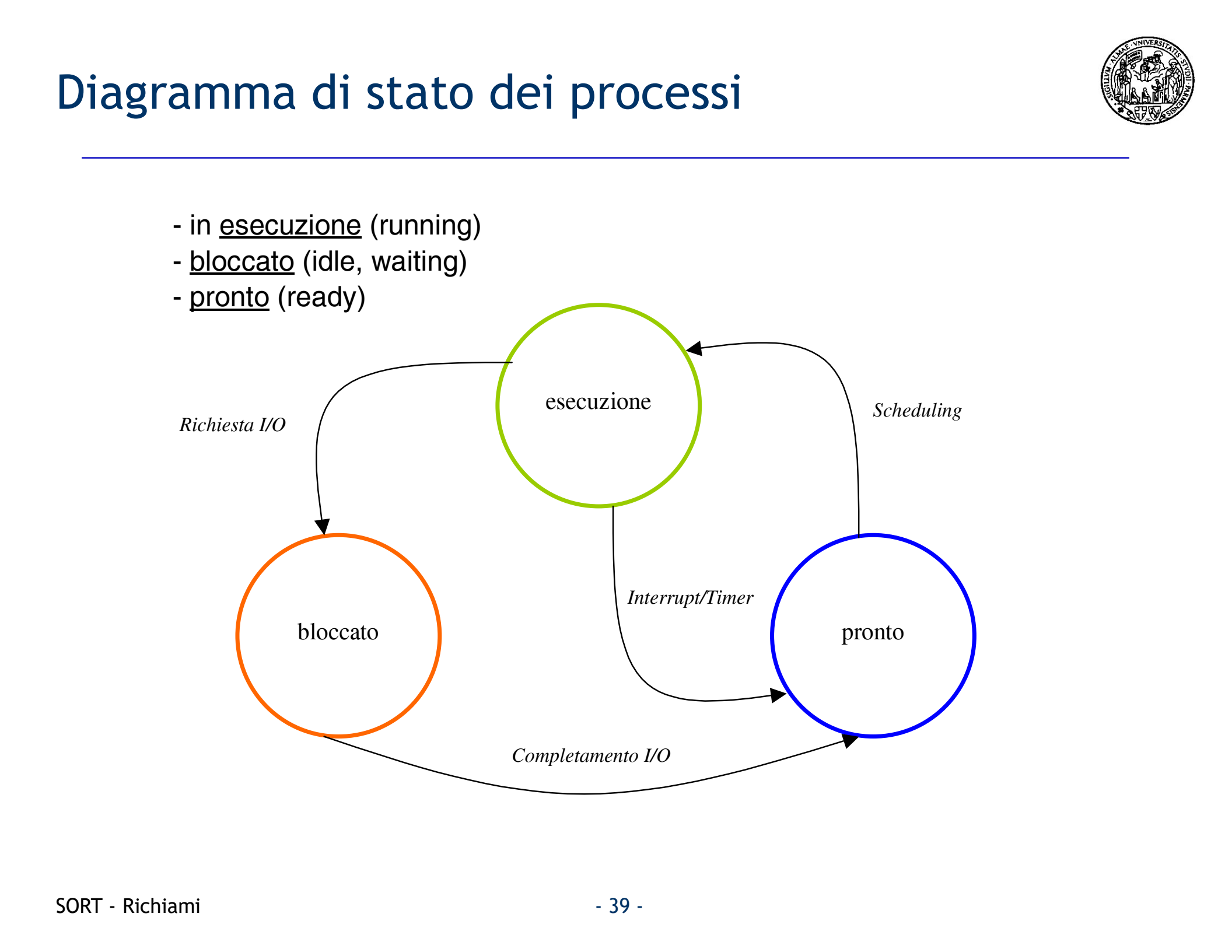
- Presenza in memoria centrale di *più programmi* in esecuzione: gestione dei programmi, protezione dalla mutua interferenza
- Il nuovo punto di vista: i programmi in esecuzione sono le entità attive; CPU, dispositivi e memoria sono risorse che vengono acquisite, rilasciate, trasferite
- Un'unica astrazione per programmi in esecuzione su CPU e attività dei processori di I/O

Processi

- Processo: programma in esecuzione
- Un sistema multiprogrammato è costituito da una collezione di processi: processi di sistema (eseguono codice del SO), processi utente
- Un processo è controllato da un programma e necessita di un processore per la sua esecuzione
- Può disporre di un processore dedicato o condividerne uno con altri

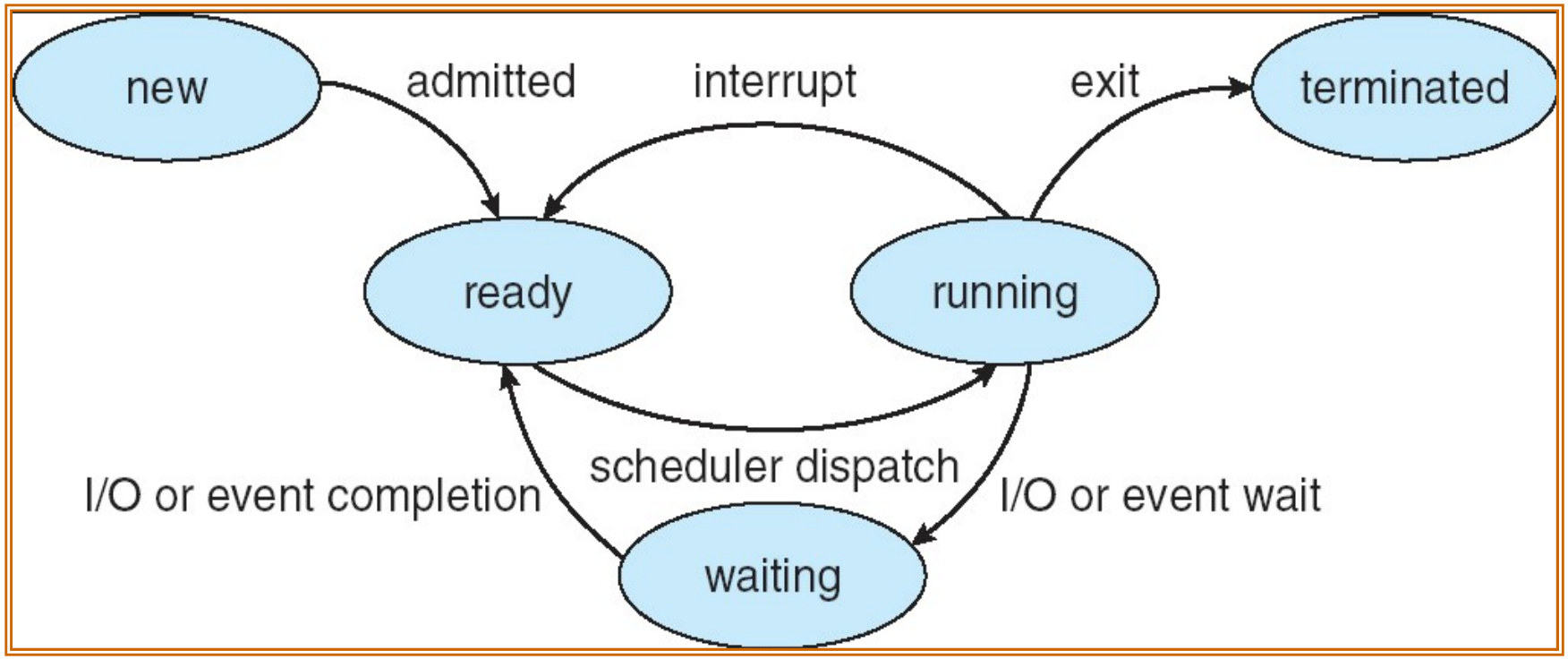
- La coesistenza di più processi in un sistema di elaborazione (ovvero la multiprogrammazione) si basa sul concetto di *stato del processo*

Diagramma di stato dei processi



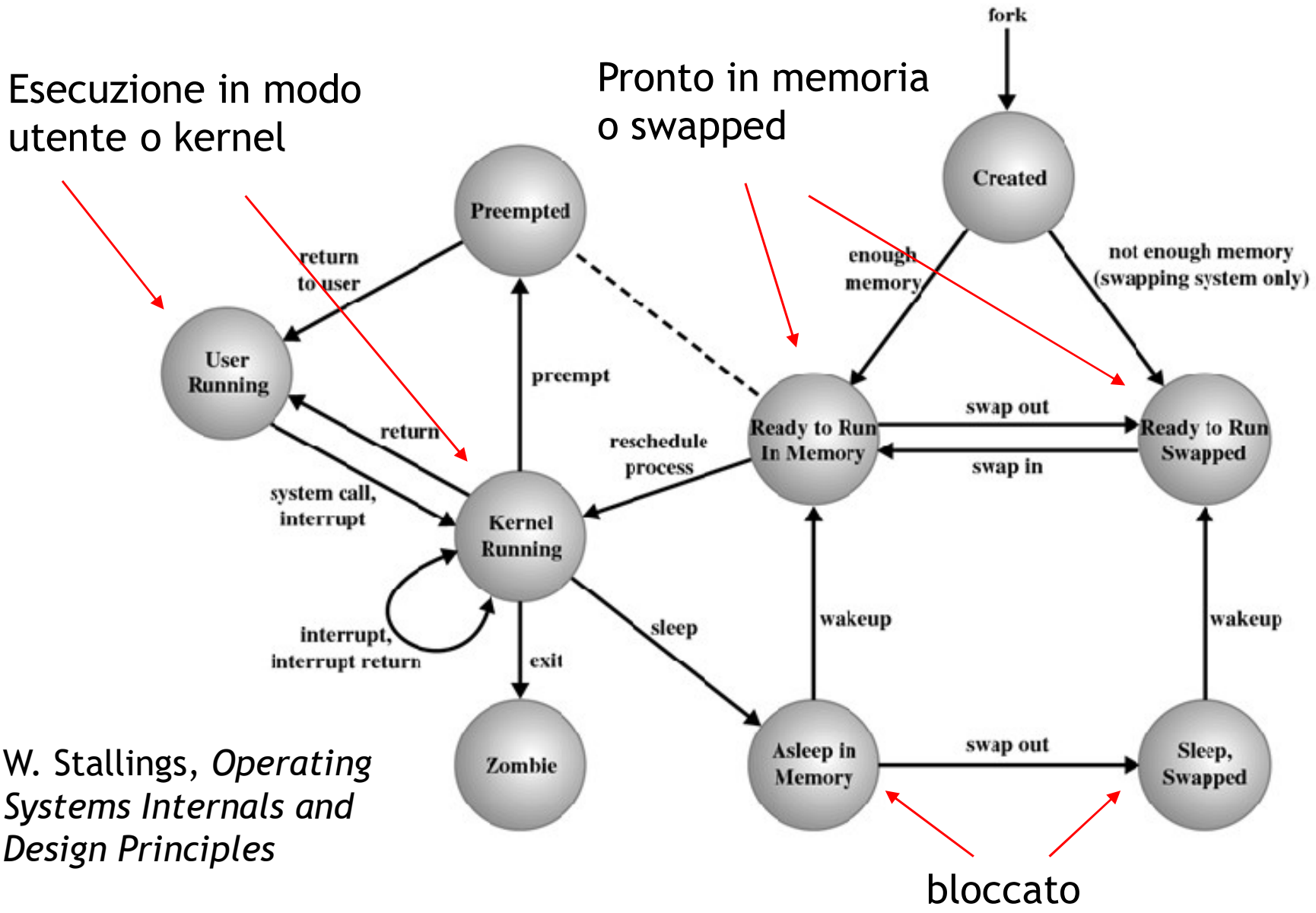
- in esecuzione (running)
- bloccato (idle, waiting)
- pronto (ready)

Diagramma di stato dei processi



New: in corso di creazione
Ready: in attesa di eseguire
Running: in esecuzione
Waiting: in attesa di qualche evento
Terminated: completati, liberano le risorse

Diagramma di stato dei processi



W. Stallings, *Operating Systems Internals and Design Principles*

Stati dei processi

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

There is no such thing as a free lunch

- TANSTAAFL --> Engineering lesson!

- TANSTAAFL indicates an acknowledgement that in reality a person or a society cannot get "something for nothing"
- Even if something appears to be free, there is *always a cost* to the person or to society as a whole, although that may be a *hidden cost* or an *externality*
- For example a bar offering a free lunch will likely charge more for its drinks

- (source: wikipedia)